Annales Univ. Sci. Budapest., Sect. Comp. 37 (2012) 5–18

ADDRESS STANDARDIZATION

Roberto Giachetta, Tibor Gregorics,
Zoltán Istenes and Sándor Sike
(Budapest, Hungary)

Communicated by Zoltán Horváth

(Received December 23, 2011; revised March 5, 2012;
accepted March 10, 2012)

Abstract. This paper is about matching non-standard format, mistyped addresses against a reference address database. The input addresses are collected from several different, non-standardized, non-verified, erroneous human input. The objective of the process is to "clean" the input address and find it in a standardized address database, or to find the most probable corresponding addresses with their appropriate credibilities. This process is known in the literature as "address cleansing".
We developed an algorithm, which searches and matches the input address fields against the standard address database stepwise, using a rule based system. Our rule based system uses tokens obtained from a specialized tokenization and generates intermediate format addresses, by identifying the missing address field and their values. The rules are grouped into rule sets and applied according to a recognition order. The tokens are matched against the address reference database using a modified Levenshtein distance measure. Our uncertainty calculus uses the modified Levenshtein distance measure matching value and the rule strength to modify with each rule application the intermediate format address credibility. Our rule base is specialized to work with Hungarian addresses, using the standard reference database.

Key words and phrases: Address cleansing, Levenshtein distance, reference address database, rule based system, search, credibility, tokenization, learning.
2010 Mathematics Subject Classification: 68T20, 68T50, 68T05.
1998 CR Categories and Descriptors: H.3.3, J.1.
The Research is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TÁMOP 4.2.1./B-09/1/KMR-2010-0003).

1. Introduction

Companies contact clients by agents. An agent visits several clients and collect data and information relevant for the company. In this process the agent records the address of the client and provides this record for further processing. The first step of this process is digitization, the second step is to identify the address in a reference database.

This article focuses on the problem of retrieving a digitized address from a reference address database. We are not concerned about the digitization process, i.e., how to transform a handwritten text to computer representation, but we are interested in matching the input address against known addresses. The source of the input address is an agent, therefore the form of the input (the granularity of the address, the order of present fields) differs, meaning that the same address can be written in different manners and fields can be missing, or due to human error the input can contain mistakes.

Addresses stored in the reference database have a well-defined structure: postal code, name of the settlement, name of the public place, type of the public place (street, road, square etc.), building number, and additional data like level, door number. These structural elements are called *fields*. Unfortunately, agents do not follow the structure described, when they collect addresses.

Our objective is to find addresses from the reference database that possibly match the input recorded, and also order and filter the results based on their credibility. In practice this is a batch process supervised by a human operator, where a large number of agent records should be processed.

The complexity of our problem arises from the following facts:

- The fields of the address are recorded in different order. Therefore, splitting the input into parts – called *tokens* – does not result in identifying fields, since a field can match more than one token. (When two strings are identified as names, it is not clear which one is the settlement and which one is the public place name, especially if both names refer to a city.)
- Fields may be missing or extra information may be added to the input. For example, large cities have districts and the district is used in the input, but it is not present in the standard.
- Fields may be abbreviated in the input.
- Fields may be mistyped, i.e., some characters are changed, missing or duplicated, or extra characters are recorded.

- There are different variants of a field's name, and any of them may be present in the input.
- The form of the building number may differ from the standard, i.e., 4–8 is used in the reference standard, but only 4 is recorded in the input.
- The form of the additional data is free, their order is arbitrary.
- The fields of the input are not consistent, e.g., the district number does not correspond to the postal code.

The problems above imply that even if the input is split into tokens using a lexical analysis, matching the tokens against possible fields is not a straightforward task [2]. When the input is split into tokens it is not guaranteed that each token will correspond to a field (additional data, or street names with more than one substring), thus tokens should be joined or split further in the matching process.

The above cleaning, matching process is called "address cleansing" [5]. Several companies offer commercial solutions, services, software for this problem, for example Cleanse+ service[1], QAS Batch address correction software[2] or the New Zealand Post[3], but they are not publicly documented.

Our objective was to develop an address standardization method and a largely customizable application prototype.

2. The concept of address standardization

Addresses are identified in multiple steps in which they are separated to several parts, called *tokens*. Separators can be whitespace characters, commas, semicolons or several special cases (e.g. when a digit is next to a letter). The process transforms these tokens to become address *fields*. An address field is a part of the address, which identifies an attribute (postal code, name of settlement, name of public place, type of public place, house number, building, staircase, floor, door) and its value. For example, the `<Bpest>` token can be recognized as a settlement name with the probable ,,Budapest" value. It is also possible that multiple adjoining tokens form a field; for instance, the `<XIV><ker>` token pair identifies the ,,14th district".

[1] http://www.postcodeanywhere.co.uk/address-cleansing
[2] http://www.qas.com/address-correction-software.htm
[3] http://www.nzpost.co.nz/business/sending-within-nz/over-300-letters-documents/address-certification/cleansing-addresses

During the process a *rule* is applied in every step [9]. Rules identify a single address field using one or more adjacent tokens. After the application of the first rule the address becomes partly identified. This state is called the *intermediate form*. The intermediate form stores the identified fields together with their values and the rule applications that have led to this form.

The intermediate form also stores a *credibility value* starting from one that is the uncertainty value of the form being the correct standardization of the original input. Every rule application has a uncertainty value (ranged between 0 and 1) based on the transformation made, and on each application of a rule the credibility of the intermediate form is multiplied by this value. Thus each step reduces the credibility of the address.

All rules have a `precondition` → `affect` format. The precondition is a token pattern, that is a sequence of token schemata. A token scheme is an algorithm that checks the compliance of a token. Table 1 shows some examples of the actually existing token schemes.

scheme name	scheme description
`<IsSettlement>`	searching in the reference database the best matching *MatchCount* number settlement names, and returns the MatchCount best matching settlement names together with their similarity score
`<IsPostalCode>`	matching to a four-digit number (Hungarian postal codes are four-digit numbers)
`<IsPostalCode @Settlement>`	matching to a four-digit number, returns 1 if the postal number corresponds (exists) in a previously identified settlement
`<IsDistrict @PostalCode>`	matches the token as a district having value [1-23] (Budapest has 23 districts) in Arabic or Roman numbers, then verify the match of the district with the postal code.
`<IsPublicPremisesName @Settlement>`	searching for the given settlement the best matching *MatchCount* number public premises name, and returns them with their Levenshtein distance metric

Table 1: Token schemes

If a pattern can be matched with a part of token sequences, then the rule can be applied. For instance, the `<IsSettlement>` scheme fits in any token that strongly resembles settlement names stored in the reference database. Another example is the `<PostalCode>` scheme, which fits numbers that have three or four digits. Rules are grouped into a rule database table according to the

address fields they recognize. An example of rule application can be seen in Figure 1.

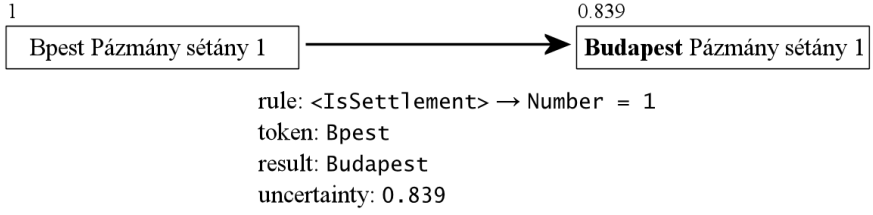


Figure 1: Rule application

Since multiple rules, moreover multiple variations of a single rule can be applied to an address at the same time, the transformation process can have several branches. The result of rule applications is a tree structure, and a branch ends when no further rule can be applied. The leaf elements of this tree are called *final addresses*. For example, the address ,,Szeged, Makó utca 5." means ,,5th Makó str. Szeged"; the `<IsSettlement>` scheme can fit in either ,,Szeged" or ,,Makó" and even to ,,utca", but with less uncertainty. The goal is to find a set of branches – also called *rule application chains* – that leads to final addresses with good credibility values.

2.1. Weighting of nodes

Every intermediate form has a credibility value, noted C. This represents how the recognitions and identifications all correspond to the intention of descriptor of the initial address. It is not a probability measure, but it is always between 0 and 1. The absolutely credible intermediate form has a value of 1, and the completely incredible has 0. The initial input address also has the value of 1, and it is decreased at every rule application that does not make a completely credible recognition.

Every rule application has an uncertainty measure that is used to multiply the intermediate credibility of form.

$$C(new\ intermediate\ form) = C(rule\ application) \cdot C(old\ intermediate\ form)$$

The credibility of the rule application depends on two factors.

1. The *strength value* of the rule, which is stored in the database, indicates the usefulness of the rule. For example, the `<IsPostalCode>` → `PostalCode = 1` with `strength = 50` rule refers to a 50% credibility

that a four digit number is a postal code, therefore the credibility of the current intermediate form is halved when this rule has been applied. The strength value can be simply updated manually to optimize rule applications and their effects.

2. The *matching value* defines the degree in which the rule fits the tokens in the concrete case. For instance, in the case of the `<IsSettlement>` → `Number = 1` rule, the difference between the current token and the settlement names stored inside the database is calculated. Candidates are chosen from the best fitting names and the better the match, the better the fitting value. The value is calculated using a similarity measure function.

The credibility of the rule application is the product of the two factors:

$$C(\textit{rule application}) = C(\textit{rule strength}) \cdot C(\textit{matching value})$$

2.2. Similarity measure

The similarity measure assigns values between 0 and 1 for two strings. The more the two strings are matching, the higher value is assigned. This function uses a modified version of the *Levenshtein-distance algorithm* with dynamic programming [4]. Compared to the original algorithm, more emphasis is put on the correspondence of the first characters and beside the possibility to insert or remove characters our version also offers character replacement. The similarity value is calculated by $sim(x, y) := \exp(-d(x, y))$, where $d(x, y)$ is the modified Levenshtein-distance.

Penalty points are calculated during every transformation of the string [2]. These penalty points vary depending on the performed action and also previous actions (e.g. three successive insertions result in less penalty points than an insertion, a removal and a second insertion). After the algorithm is completed, the penalty points are transformed into interval $[0, 1]$.

3. The problem space

The problem space of the address recognition algorithm can be outlined as a directed graph. The vertices of the graph represent intermediate forms and the root vertex corresponds to the original input address that has no identified fields. The leaf vertices are intermediate forms where no further rule can be applied either because all tokens have been identified, or all possible rules have been applied. The leaf vertices are the final address candidates. Edges represent the rule applications. The graph does not contain any circles, because no rule reduces the number of identified tokens.

One vertex has several children since several rules can be applied to one intermediate form, moreover one rule can produce several results. For example (see Figure 2), the settlement identification rule maps from the string ,,Gár" to ,,Gárdony", to ,,Gyál" and to several others. Even the vertex ,,Gárdony 2483 I. u. 9" on the second level has got several children because the ,,postal code" rule can be also matched to the number 9 and the ,,empty" rule (it means that there is no postal code) may be applied. Certainly, the credibility values of these children are worse than 0.75.

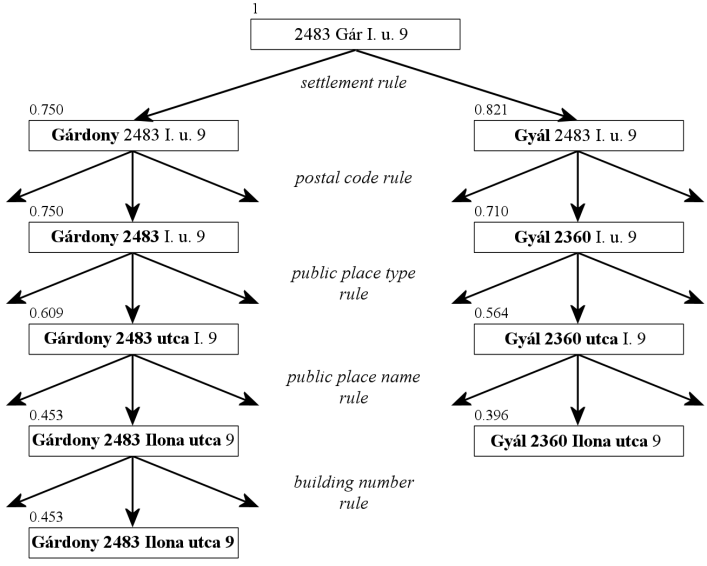


Figure 2: Problem space segment

The size of the graph is limited by reducing the set of applicable rules. A group of rules can only be applied at a certain step of the process, which also defines the level of the graph in which it can be applied. Therefore, the rules of step (or level) $k + 1$ can only be applied after applying a rule on level k. Since the input addresses might not be complete (e.g. missing postal number or street number) an `<empty>` rule is introduced that can be applied at most levels and jumps to the next level; however it also decreases the credibility value of the form.

An example of problem space segment can be seen in Figure 2. Identified tokens are displayed in bold, and credibility values can be seen in the upper left corner of the vertex.

Using this approach, the graph is reduced to a tree structure where a path between the root and a leaf represents a rule application chain, in which the

credibility value is monotone decreasing. An effective search strategy is necessary for the handling of this problem space in acceptable running time [3].

3.1. Search strategy

Within the problem space, the task is to find leaves (these are considered to be *result vertices*) that have the highest credibility values. Leaves generally do not have unidentified tokens, however, some addresses have parts that cannot be processed based on the standard addressing (e.g. ,,the 2nd shop at garage level", ,,back in the alley"), therefore all leaves are taken into account as results. Due to the large size of the tree, the goal can only be achieved with a *best-first search* heuristic graph algorithm [7] with some modifications.

Starting from the root vertex, a vertex with the highest credibility value is selected in each step and removed from the priority queue where the queue contains vertices whose successors are not yet known. Rules are applied to this vertex and they create successors, which are put into the queue. This is repeated until a certain amount of leaf vertices are found. When unrecognized tokens remain in the address, the credibility value is reduced again. Since the optimal solution is not always the first solution, the algorithm does not terminate when it finds a leaf vertex.

A *cutting heuristic* is used to speed up the search. It only considers inner vertices whose credibility values are not worse than that of the best result found so far minus a threshold value (called *CutLimit*).

For this search, several strategic parameters have been introduced that are also stored and can be modified to fine-tune the algorithm. These parameters include:

- *Delimitations* for cutting, the size of the priority queue, maximum distance of the Levenshtein algorithm, etc. are used to reduce the size of the problem space and therefore accelerate the algorithm, but this may also result in the loss of good results. See Table 2.

- *Validation penalties* are applied when the resulting values of a rule application do not match, e.g. the postal code does not belong to the settlement, or the number does not exist in the street. See Table 3.

- *Missing field penalties* are also applied in case fields are missing in a result address, with the most important fields (settlement name, street name) earning the most penalties. See Table 4.

Name	Default	Range	Description
CutLimit	0.05	$[0, 1]$	The maximum difference allowed below the uncertainty of the best intermediate form. Intermediate forms with worse uncertainty are not enqueued in the priority queue.
PQueueSizeLimit	50	$(0...\infty)$	The maximum size of the priority queue used for intermediate form storage.
MatchCount	10	$(0...\infty)$	The maximum number of matching words returned by the Levenstein distance function.
DistanceLimit (DL)	0.4	$[0...\infty)$	A general limit value used in several cases, e.g. as the maximum distance value allowed by the Levenstein function. (A value of 3 usually refers to largely unmatching words).
ValidatedsCount-Limit	5	$[0...\infty)$	The upper limit for the number of addresses after validation.

Table 2: Delimitation parameters

Name	Default	Range	Description
PostalCode-Punishment	0.2	$(0, DL]$	Penalty used in case the postal code does not match the settlement name.
DistrictPunishment	0.2	$(0, DL]$	Penalty used in case the district number does not match the postal code.
DistrictChange-Punishment	0.21	$(0, DL]$	Penalty used when correcting the district number according to the postal code.
ValidatePunishment	0.8	$(0, 1]$	Penalty used in case of inconsistency during validation of the public place name rule.
ChangeByValidation-Punishment	0.98	$(0, 1]$	Penalty used in case of token change in an intermediate form due to inconsistency.
InvalidStreetNumber-Distance	0.05	$(0, DL]$	Penalty used when the street number does not match the rest of the address.

Table 3: Validation penalty parameters

Name	Default	Range	Description
MissingPostalCode	0.8	[0, 1]	Missing postal code.
MissingSettlement	0.5	[0, 1]	Missing settlement name.
MissingStreetName	0.7	[0, 1]	Missing public place name.
MissingStreetType	0.7	[0, 1]	Missing public place type.
MissingStreetNumber	0.7	[0, 1]	Missing street number.
MissingDistrict	0.8	[0, 1]	Missing district number.
UnknownTokenPunishment	0.8	[0, 1]	Unrecognized token.

Table 4: Missing field penalty parameters

3.2. Learning

The effectiveness of the search depends strongly on the credibility values calculated for the intermediate forms and on the strategic parameters used. These focus on the search and enable the earliest finding of the most credible results. The credibility values and the strategic parameters must be tuned carefully, since inadequate values lead to the loss of good results by rejecting intermediate forms, or penalizing good candidates. This tuning requires experience, domain knowledge and/or using trial-error methods. However, the lack of them can be replaced with a learning mechanism, which can modify these values based on the experiences of the already resolved tasks.

The usefulness of a step (or the decisive logic that generates it) can only be judged after seeing whether the step operation led us to any solution or to a good solution. In this case, all those steps that led us to a solution are reinforced proportionately [8] to the quality of the good solution.

The easiest way to achieve reinforcement is learning by taking the usefulness of the rule as the quotient of the number of the efficient uses of all the uses. For all the rules, we can tell precisely how often its use led us to successful address recognition. In favour of this, in their intermediate address forms, we record which rule was used, and based on this, it can be easily scored how many times a rule was used, and from that, how many of them led to a successful address recognition.

$$C(\textit{rule strength}) = \frac{\textit{number of the efficient uses}}{\textit{number of all the uses}}$$

When a final address is made, the number of ,,all the uses" of the applied rules can be increased. (The final address stores all the applied rules that create it.) If a final address is good (formerly we planned that the end-user can decide this, in the present application the best final address is the good one) then the number of ,,all the efficient uses" of the applied rules can be created. So the above quotient is calculated automatically.

Another way of learning might be the correction of called *strategical parameters* during the execution to influence the efficiency and effectiveness of the model.

4. The prototype application

Based on the theory described in Section 2 and 3 a prototype application has been developed[4]. The implementation was carried out using the .NET Framework infrastructure with MySQL database background.

The database consists of multiple tables and has two parts.

- The first part is the collection of standard addresses in Hungary, e.g. settlement names, public place names and also public place types (street, square, way, etc.)[5]. There is also a table that contains the renaming of public places (e.g. in 2010 the ,,Moszkva tér" in Budapest was renamed ,,Széll Kálmán tér"). The address list is accurate to the street level, but there are no data stored regarding floor and door number information in a building. For the prototype, only part of the Hungarian address list was used including the most problematic locations.

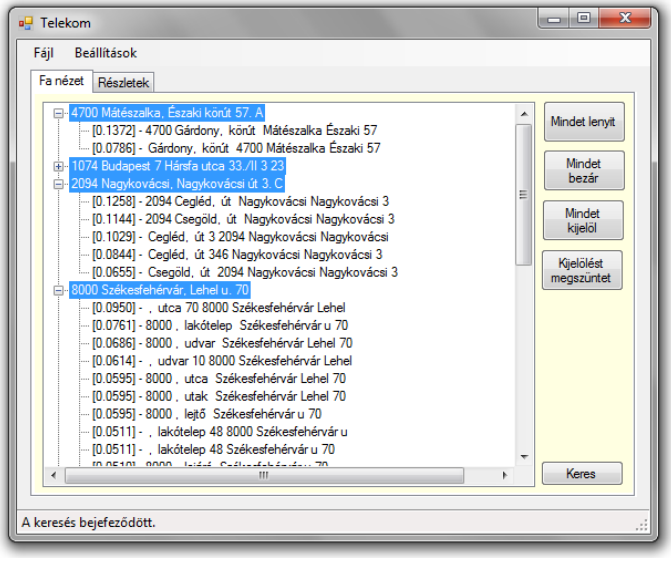
- The second part consists of the rule tables, which are build up as described in Section 2. For the prototype, 34 rules have been assembled with level values reaching up to 11. Therefore, the rule application chain has a maximum length of 11.

Strategic parameters are stored in XML configuration files, but can easily be modified from the settings dialog of the application.

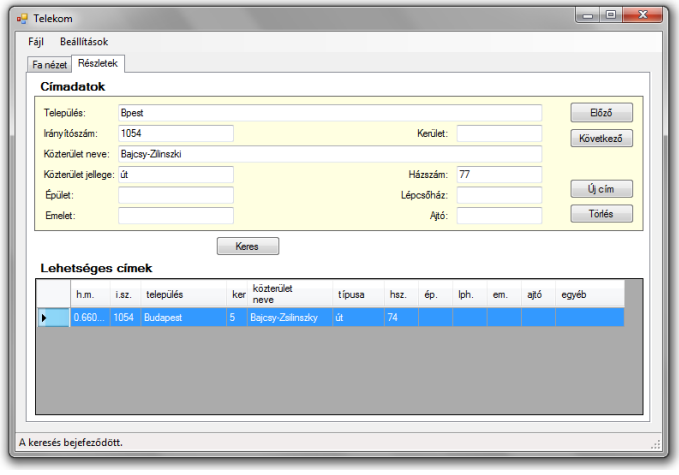
The application has a graphical user interface allowing the input of multiple addresses and gathering results according to credibility values. As can be seen in Figure 3, there are two screens. One for all results of multiple addresses, and one for detailed results of a single address. Result numbers can vary depending on strategic parameters. In both screens the credibility values can be clearly seen, thus the operator can make the final choice from the best results, or manually adjust parameters to recalibrate the process.

[4]Development was performed by students Csaba Bálint, Áron Baráth, Tamás Bibók, Csaba Czine, Eszter Ginál, Csaba Horváth and Richárd Szabó.

[5]In the prototype application the rule base was created to deal with Hungarian addresses. The application graphical interface was also created in Hungarian. That is why in this paper the examples are given mostly as Hungarian addresses. The rule base can be easily changed to comply other different address standard and reference databases.



(a) Results on multiple addresses



(b) Detailed result for a single address

Figure 3: The prototype application

5. Conclusion

We developed a prototype application to clean in a batch process non standardized, non-verified addresses and match them against a reference address database. We used a rule based system to stepwise identify the missing fields. Tokens are matched using token schemes and modified Levenshtein distance measures. Credibility values are created and updated with reinforcement learning. A prototype application specialized for the Hungarian addresses and a Hungarian reference address database[6] proved the effectiveness of our address standardization approach and system.

The prototype application created a list of candidate addresses for an input string, as the original specification of the system required. The reason for this requirement has been that previous software tools used were not able to produce the address expected by human in some cases. The test results of our prototype were much better: for each input it produced the address expected by human supervisor with the highest credibility value. This means that the requirement specified may be changed, and it would be sufficient to show only the best result and eliminate the human control.

References

[1] **Gusfield, D.,** *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, 1997.

[2] **Hall, P.A.V. and G.R. Dowling,** Approximate string matching, *ACM Comput. Surv.*, **12** (1980), 381–402.

[3] **Knuth, D.E.,** *The Art of Computer Programming, Volume 3: (2nd ed.) Sorting and Searching*, Addison Wesley Longman Publishing Co., Inc., 1998.

[4] **Levenshtein, V.I.,** Binary codes capable of correcting insertions and reversals, *Soviet Physics Doklady*, **10** (1966), 707–710.

[5] **Müller, H.,** *Problems, Methods and Challenges in Comprehensive Data Cleansing*, Technical Report, (2003).

[6] **Navarro, G.,** A guided tour to approximate string matching, *ACM Comput. Surv.*, **33** (2001), 31–88.

[6] In Hungarian http://www.geox.hu/gis-terkepek/dsm10

[7] **Russell, S.J. and P. Norvig,** *Artificial Intelligence: A Modern Approach*, Pearson Education, 2003.

[8] **Sutton, R.S. and A.G. Barto,** Reinforcement learning: An introduction, *IEEE Transactions on Neural Networks*, **9** (1998), 1054–1054.

[9] **Jackson, P.,** *Introduction to Expert Systems* (*3 ed.*), Addison Wesley, 1998, ISBN 978-0-201-87686-4.

R. Giachetta, T. Gregorics, Z. Istenes and S. Sike
Department of Programming Languages and Compilers
Faculty of Informatics
Eötvös Loránd University
H-1117 Budapest, Pázmány P. sétány 1/C
Hungary
groberto@inf.elte.hu
gt@inf.elte.hu
istenes@inf.elte.hu
sike@inf.elte.hu